МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Системного программирования»

Дисциплина «Операционные системы»

Отчет по курсовой работе

Выполнил:

студент группы БСТ2104

Мажукин И.Н.

Проверила: Алексанян Д. А.

Москва, 2023 г.

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Разработать многозадачное приложение, выполняющее получение, сбор и вывод системной информации в соответствии с вариантом задания (таблица 1). Номер варианта задания на курсовую работу совпадает с номером, под которым записана фамилия студента в журнале группы.

Таблица 1 – Индивидуальный вариант задания

|  |  |  |
| --- | --- | --- |
| Вариант | Сервер 1 | Сервер 2 |
| получает от клиента/выполняет и возвращает в качестве результата | |
| 7 | * название используемого видеоадаптера * размер клиентской области | * количество свободных байтов файла подкачки * объем используемой физической памяти в единицах, переданных клиентом (в байтах, мегабайтах, гигабайтах по выбору пользователя) |

**Дополнительные задания**

1. Реализация графического интерфейса на стороне клиента.
2. При периодическом обновлении информации по запросу от клиента (организовать выбор режима отправки периодических запросов к серверу в рамках текущей сессии) новые данные передается сервером в ответ только в том случае, если что-либо изменилось с момента прошлого обновления информации для данного клиента.
3. Наличие версии сервера или клиента для другой ОС (т.е. если клиент и сервер могут работать под разными ОС).

**СОДЕРЖАНИЕ**

[**1. Введение** 4](#_Toc154089298)

[**2.** **Постановка задачи** 5](#_Toc154089299)

[**3. Теоретические основы разрабатываемой темы** 6](#_Toc154089300)

[**4. Разработка и описание алгоритма работы приложения** 7](#_Toc154089301)

[**5. Результат работы программы** 10](#_Toc154089302)

[**6. Заключение** 12](#_Toc154089303)

[**Список использованных источников** 13](#_Toc154089304)

[**Приложения** 14](#_Toc154089305)

# **Введение**

В современном мире, где технологии проникают во все сферы нашей жизни, передача и обработка данных становится неотъемлемой частью нашей повседневной деятельности. От промышленных комплексов до систем автоматизации торговли — все они тесно переплетены с необходимостью эффективного обмена информацией.

В этом контексте использование сокетов в клиент-серверных приложениях представляет собой элегантное и востребованное решение. Эти приложения выступают в роли моста, обеспечивающего постоянное взаимодействие между различными процессами. Сервер, выступая поставщиком услуг, готов ожидать запросов от клиентов, предоставляя им необходимые данные или решения вычислительных задач. Клиент, в свою очередь, будучи инициатором запроса, обладает гибкостью подключаться к серверу в любой момент, повышая тем самым эффективность обмена информацией.

Цельюданной курсовой работы является закрепление и углубление теоретических знаний в области современных операционных систем, приобретение практических навыков разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

# **Постановка задачи**

Основной задачей данной курсовой работы является разработка многозадачного приложения, выполняющего получение, сбор и вывод системной информации в соответствии с индивидуальным вариантом задания.

Многозадачное приложение реализуется на основе технологии «клиент-сервер» средствами интегрированной среды разработки Visual Studio Code на языке программирования Python3. Процессы 1 и 2 выступают как серверы приложений, третий процесс является их клиентом. Обмен данными между клиентами и серверами осуществляется при помощи сокетов.

Многозадачное приложение реализуется на основе технологии «клиент-сервер». Два сервера должны поддерживать многопоточность, т.е. обеспечивать одновременное подключение нескольких клиентов. Каждый сервер должен обрабатывать запросы клиентов, предоставляя системную информацию в соответствии с вариантом задания, и передавать её клиенту с указанием времени. Также необходимо разработать клиентскую часть с интерфейсом, позволяющим выбирать сервер для подключения, с возможностью раздельного получения и отображения данных по каждому из пунктов индивидуального задания.

# **3. Теоретические основы разрабатываемой темы**

1. Архитектура «клиент-сервер»:

Архитектура «клиент-сервер» - это модель взаимодействия компьютерных сетей, при которой на одном конце соединения находится клиент, а на другом - сервер. Сервер предоставляет услуги или ресурсы, а клиент их запрашивает.

**Плюсы:**

* **Эффективность**: архитектура «клиент-сервер» позволяет эффективно использовать ресурсы сервера, так как клиент может быть бездействующим, пока сервер обрабатывает запрос.
* **Масштабируемость**: архитектура «клиент-сервер» легко масштабируется, так как можно добавить больше серверов для обработки запросов от большего числа клиентов.
* **Гибкость**: архитектура «клиент-сервер» позволяет использовать различные протоколы и технологии для обмена данными между клиентом и сервером, что делает ее гибкой для различных приложений.

**Минусы**:

* **Надежность**: если сервер не работает, все клиенты не смогут получить доступ к ресурсам или услугам, которые предоставляет сервер.
* **Безопасность**: архитектура «клиент-сервер» может быть уязвима для атак, так как клиент может отправлять запросы на сервер, которые могут содержать вредоносный код или попытки взлома.

2. Системные средства коммуникации и синхронизации процессов

* **Сокеты.** Сокет — это средство связи, позволяющее разрабатывать «клиент-сервер» системы для локального или сетевого использования.
* **Многопоточность**: архитектура «клиент-сервер» позволяет обрабатывать несколько клиентских запросов одновременно, используя отдельные потоки на сервере.
* **Протоколы передачи данных**: для обмена данными между клиентом и сервером используются протоколы TCP/IP, которые обеспечивают надежность и управление соединением (TCP) и маршрутизацию и доставку пакетов данных (IP).

# **4. Разработка и описание алгоритма работы приложения**

«Клиент-сервер» система состоит из 5 файлов расширением .py: первый сервер (server\_1), второй сервер (server\_2), клиент (client), файлы для графического изображения серверов (server\_design) и для клиента (client\_desing). Первый и второй серверы имеют один и тот же HOST = "192.168.1.7", но разные порты (для сервера 1 PORT = 2233, для сервера 2 PORT = 2234). После запуска серверов, они ожидают входящее соединение.

После установления соединения клиент выбирает сервер, к которому он хочет подключиться и при этом клиент, отправляя запрос на выбранный сервер с помощью сокетов, возвращает на экран данные от выбранного сервера.

Рассмотрим теперь запросы, которые может пользователь запросить от каждого сервера:

Сервер 1:

* **Вывод название используемого видеоадаптера:**

Клиент, подключившись к первому серверу, передаёт названия видеокарт.

* **Размер клиентской области:** Клиент, подключившись к первому серверу, передаёт размер клиентской области.

Сервер 2:

* **Количество свободных байтов файла подкачки:** Клиент, подключившись ко второму серверу, получает информацию о количестве свободных байтов файла подкачки сервера.
* **объем используемой физической памяти в единицах, переданных клиентом (в байтах, мегабайтах, гигабайтах по выбору пользователя):** Клиент, подключившись ко второму серверу, получает информацию об объеме используемой физической памяти сервера в единицах, которые пользователь выбирает из списка в графическом окне.

Рассмотрим теперь модули, функции и классы, которые были использованы при разработке серверов и клиентов:

**Модули**:

Для сервера 1 и 2:

* **socket** – используется для создания сетевого соединения и обмена данными между клиентами и сервером;
* **threading** – применяется для организации многозадачности и обработки каждого клиента в отдельном потоке;
* **sys** – позволяет управлять системными ресурсами, работать с аргументами командной строки и другими компонентами, связанными с выполнением программы;
* **os** – предоставляет функционал для взаимодействия с операционной системой, например, определения приоритета процесса.
* **datetime** - позволяет выполнять операции с датами и временными интервалами, форматировать дату и время, извлекать информацию о текущей дате и времени.
* **PyQt5.QtCore, PyQt5.QtWidgets, PyQt5.QtGui** - модули являются частью фреймворка PyQt5, который предоставляет возможности для создания графического пользовательского интерфейса (GUI) в приложениях Python. PyQt5.QtCore содержит основные классы и функции для работы с событиями, таймерами и потоками. PyQt5.QtWidgets предоставляет готовые виджеты для построения UI, такие как кнопки, текстовые поля, окна и диалоги. PyQt5.
* **server\_desing** - пользовательский модуль содержит класс Ui\_MainWindow, который определяет графический интерфейс главного окна сервера приложения.

Для клиента:

* **socket** – используется для создания сетевого соединения и обмена данными между клиентами и сервером;
* **threading** – применяется для организации многозадачности и обработки каждого клиента в отдельном потоке;
* **sys** – позволяет управлять системными ресурсами, работать с аргументами командной строки и другими компонентами, связанными с выполнением программы;
* **time** - позволяет выполнять операции с датами и временными интервалами, форматировать дату и время, извлекать информацию о текущей дате и времени.
* **PyQt5.QtWidgets** - модуль является частью фреймворка PyQt5, который предоставляет возможности для создания графического пользовательского интерфейса (GUI) в приложениях Python. PyQt5.QtWidgets предоставляет готовые виджеты для построения UI, такие как кнопки, текстовые поля, окна и диалоги. PyQt5.
* **client\_desing** - пользовательский модуль содержит класс Ui\_MainWindow, который определяет графический интерфейс главного окна сервера приложения
* **enum -** Модуль enum содержит в себе тип для перечисления значений с возможностью итерирования и сравнения**.**

**Функции:**

Для сервера 1 и 2:

ServerThread - это класс-поток, который расширяет функциональность базового класса Thread. Он служит для создания отдельного потока, в котором будет выполняться серверная логика.

\_\_init\_\_(self, window) - конструктор класса, принимающий окно (window) в качестве параметра.

run(self) - метод запускается при вызове start() для потока. Внутри этого метода создается серверный сокет, привязывается к определенному хосту и порту (HOST и PORT), ожидает подключения клиентов и для каждого нового клиента запускает обработку подключения в новом потоке с помощью handle\_connection.

run() содержит основную логику сервера - устанавливается соединение через сокет, привязывается к определенному хосту и порту.

Ожидает подключения клиентов с помощью accept().

При каждом новом подключении создается экземпляр потока (Thread) для обработки этого подключения функцией handle\_connection.

Обработка подключения осуществляется в отдельном потоке для каждого клиента.

handle\_connection(sock, addr, window) - это функция, обрабатывающая подключение клиента:

sock - сокет, связанный с клиентом.

addr - адрес клиента.

window - окно GUI, используемое для отображения информации.

Для сервера клиента:

* **Функция sendToServer(self, data, serverType)** предназначена для отправки данных на сервер. Параметры функции: self: ссылка на текущий объект. data: данные, которые нужно отправить на сервер. serverType: целое число, указывающее тип сервера (0 или 1).
* **Функция runClient(serverType)** предназначена для запуска клиента и установления соединения с сервером. Параметры функции: serverType: строка, указывающая тип сервера (например, "http" или "ftp").
* **receiveFunc(serverType).** предназначена для получения сообщений от сервера. Параметры функции: serverType: строка, указывающая тип сервера (например, "http" или "ftp").
* **runClient(serverType)** предназначена для запуска клиента и установления соединения с сервером. Параметры функции: serverType: строка, указывающая тип сервера (например, "http" или "ftp").
* **check\_server(ip, port)**. предназначена для проверки доступности сервера по указанному IP-адресу и порту. Параметры функции: ip: IP-адрес сервера, к которому выполняется проверка доступности. port: порт сервера, к которому выполняется проверка доступности.
* **send\_coords\_Button()** кнопка для обработки задачи по смене коррдинат серверного окна.
* **send\_names\_Button()** кнопка для обработки задачи по выводу имени компьютера и пользователя
* **get\_priority\_Button()** кнопка для обработки задачи по выводу PID серверного процесса.
* **get\_thread\_id\_Button()** кнопка для обработки задачи по выводу PID потоков серверного процесса.
* **addItem(self, serverType, data)** предназначена для вывода текста в графический интерфейс. Параметры функции: self: ссылка на текущий объект. serverType: целое число, указывающее тип сервера (0 или 1). data: текстовая строка, которую необходимо вывести.

# **5. Результат работы программы**

Результаты работы приложения представлены на рисунках 1-9:

Запускаем клиент в терминале VSC. Результат представлен на рисунке 1

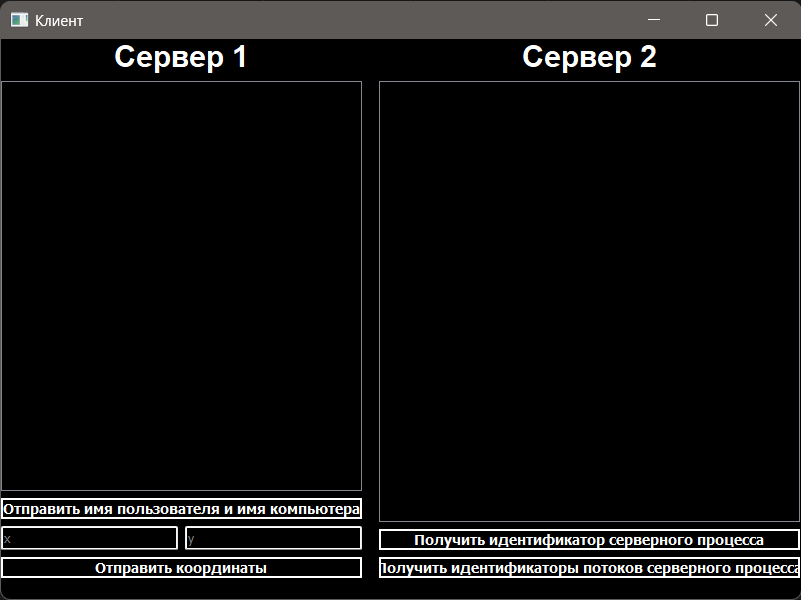


Рисунок 1 – Запуск клиента

Запускаем сервер 1 и выводим информацию об именах пользователя и компьютера

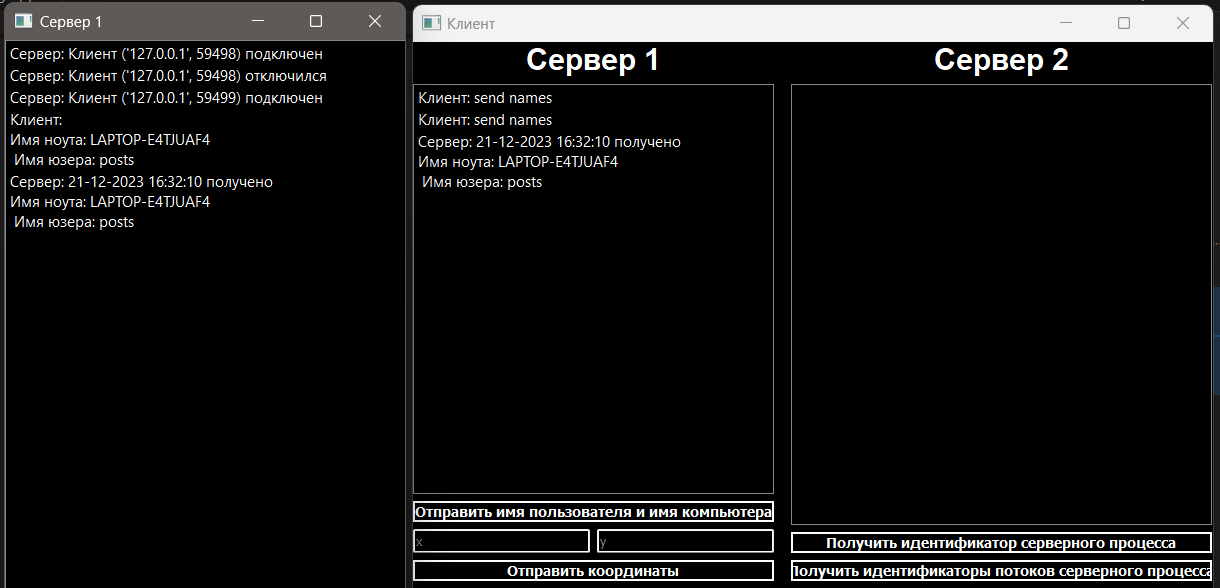


Рисунок 2 – Запуск сервера 1 и вывод необходимой информации

Переместим окно сервера по координатам 0x0 и проверим работу программы.

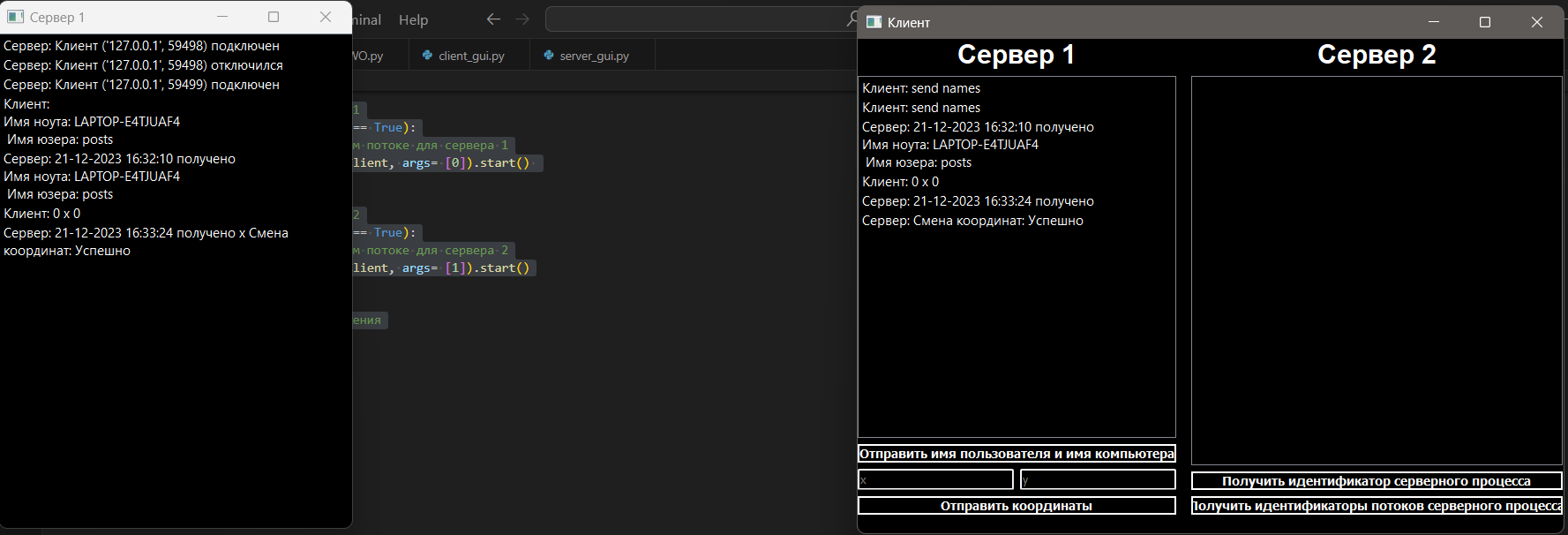


Рисунок 3 – Изменение координат положения окна сервера 1

Попробуем ввести неправильные координаты, например для X: ааа, для Y: ббб.

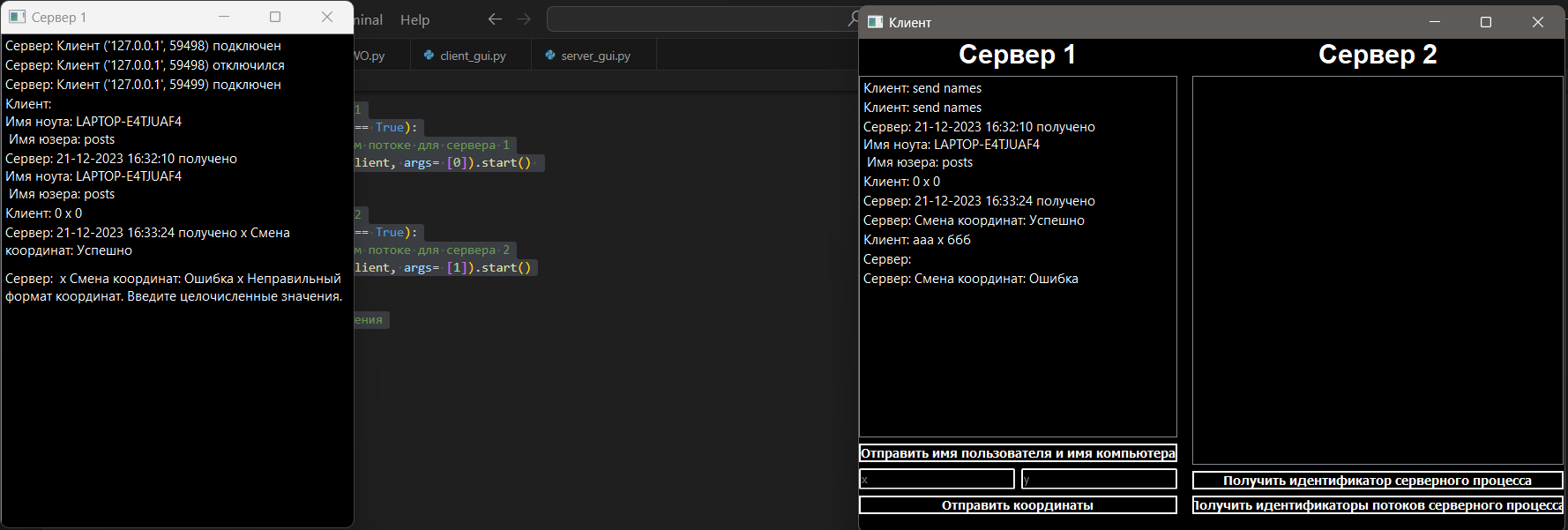


Рисунок 4 – Изменение координат положения окна сервера 1

Запускаем сервер 2 выводим PID серверного процесса. Для проверки используем PoweShell используя команду tasklist | findstr "python.exe"

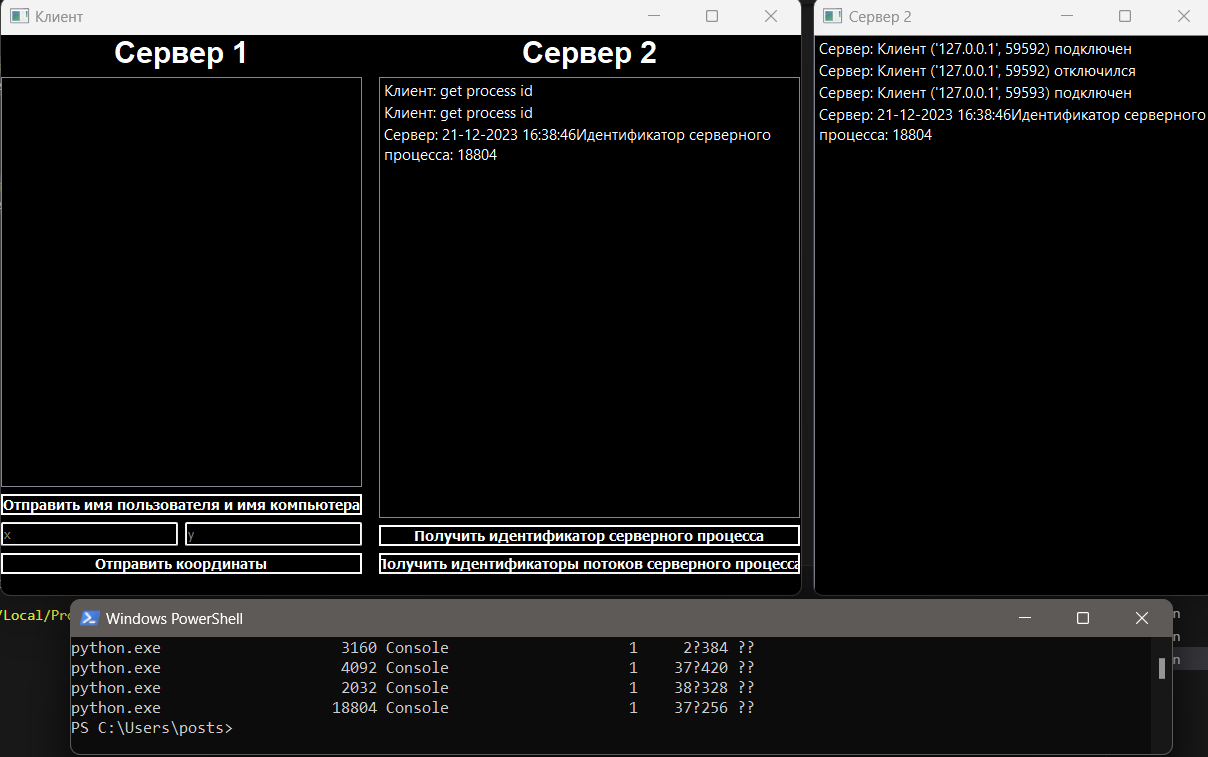


Рисунок 4 – Запуск сервера 2 и вывод необходимой информации

Запускаем сервер 2 выводим PID серверного процесса. Для проверки используем PoweShell используя команду Get-Process -Id PID | Select-Object -ExpandProperty Threads | Select-Object Id

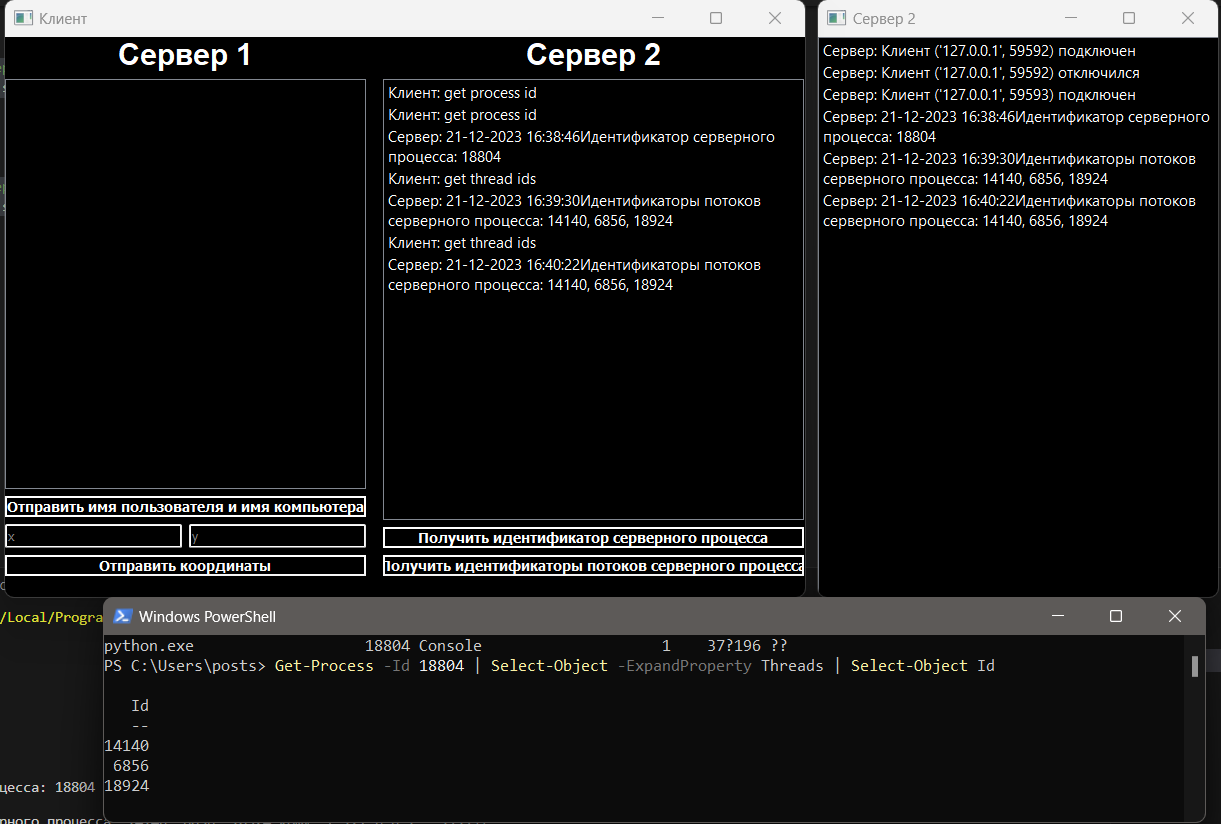


Рисунок 4 – Вывод необходимой информации

# **6. Заключение**

В результате выполнения данной курсовой работы было разработано клиент-серверное приложение, выполняющее получение, сбор и вывод системной информации» в соответствии с индивидуальным вариантом задания. В ходе работы также были закреплены и углублены теоретические знания в области современных операционных систем, приобретены практические навыки разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

# **Список использованных источников**

1. Э.Таненбаум. Современные операционные системы. 3-изд. – СПб.: Питер, 2011
2. Э.Таненбаум, Х.Бос. Современные операционные системы. 4-изд. – СПб.: Питер, 2015
3. В.Г.Олифер, Н.А.Олифер. Сетевые операционные системы. – СПб.: Питер, 2009
4. С.В. Назаров, А.И. Широков. «Современные операционные системы». — М., Бином, 2013
5. Марк Руссинович, Дэвид Соломон, Алекс Ионеску, Павел Йосифович. Внутреннее устройство Windows.7-е издание . - СПб.: Питер, 2018
6. Аллан Фарингтон «Windows 11: полное руководство пользователя новой операционной системы». — Издательский отдел факультета ВМиК МГУ, 2008

# **Приложения**

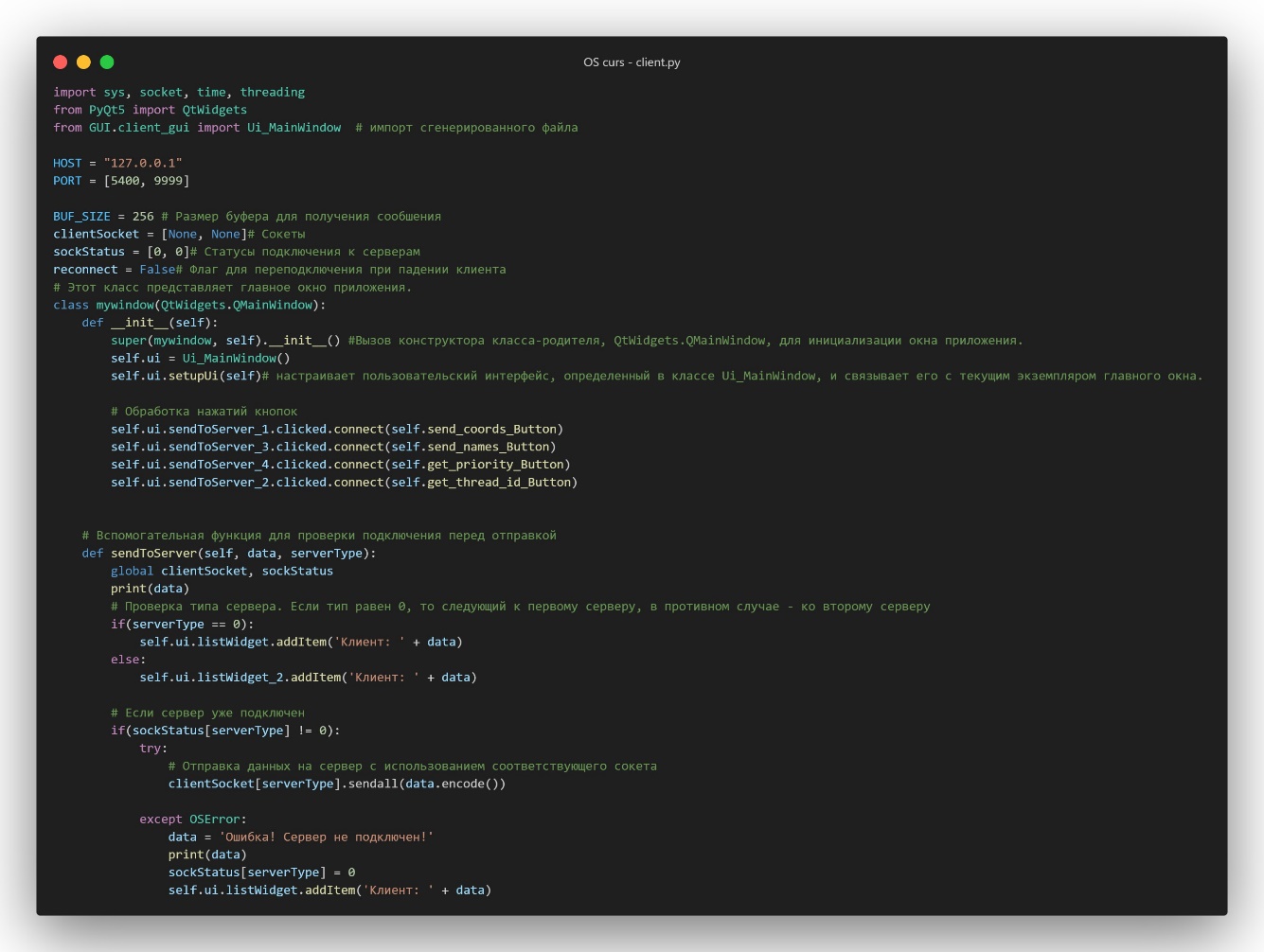


Рисунок 5 – Код Клиента



Рисунок 6 – Код Клиента



Рисунок 7 – Код Клиента

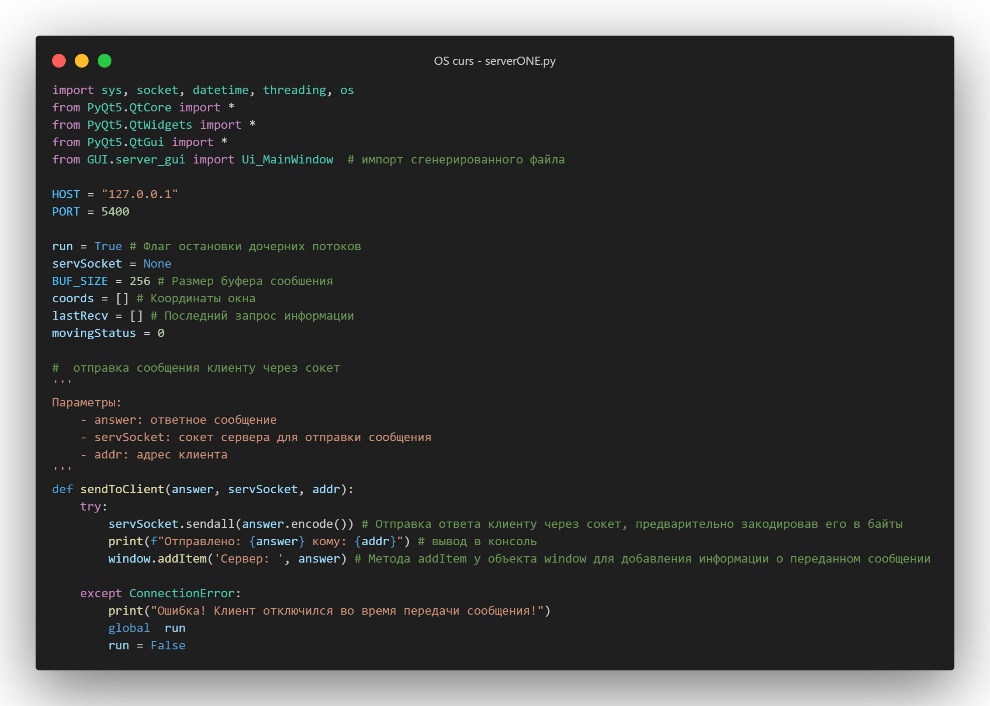


Рисунок 8 – Код Сервера 1



Рисунок 9 – Код Сервера 1



Рисунок 10 – Код Сервера 1



Рисунок 11 – Код Сервера 1



Рисунок 12 – Код Сервера 2



Рисунок 13 – Код Сервера 2



Рисунок 14 – Код клиентского окна



Рисунок 15 – Код клиентского окна



Рисунок 16 – Код серверного окна